MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Simulation-based inference for plan monitoring

Neal Lesh, James Allen

TR99-06 December 1999

Abstract

The dynamic execution of plans in uncertain domains requires the ability to infer likely current and future world states from past observations. This task can be cast as inference on Dynamic Belief Networks (DBNs) but the resulting networks are difficult to solve with exact methods. We investigate and extend simulation algorithms for approximate inference on Bayesian networks and a propose a new algorithm, called Rewind/Replay, for generating a set of simulations weighted by their likelihood given past observations. We validate our algorithm on a DBN containing thousands of variables, which models the spread of wildfire.

Proceedings of the 16th National Conference on Artificial Intelligence, Orlando, Florida, 1999

MERL – A MITSUBISHI ELECTRIC RESEARCH LABORATORY
http://www.merl.com

Simulation-based inference for plan monitoring

Neal Lesh James Allen*

TR-99-06 February 1999

Abstract

The dynamic execution of plans in uncertain domains requires the ability to infer likely current and future world states from past observations. This task can be cast as inference on Dynamic Belief Networks (DBNs) but the resulting networks are difficult to solve with exact methods. We investigate and extend simulation algorithms for approximate inference on Bayesian networks and a propose a new algorithm, called Rewind/Replay, for generating a set of simulations weighted by their likelihood given past observations. We validate our algorithm on a DBN containing thousands of variables, which models the spread of wildfire.

Submitted to AAAI'99.

*University of Rochester

Publication History:–

1. First printing, TR-99-06, February 1999

1 Introduction

In domains that contain uncertainty, evidential reasoning can play an important role in plan execution. For example, suppose we are executing an evacuation plan and have received the following messages:

> Bus_1: Arrived in Abyss, loading 5 passengers.
> Exodus weather: storm clouds are forming to the east.
> Bus_2: Engined overheated on way to Delta.
> Bus_1: Got flat tire on way to Barnacle.
> Bus_2: Loading 9 passengers.
> Bus_1: It is starting to snow in Barnacle.
> Bus_3: Got flat tire on way to Calypso.

Given the messages received so far, we might ask questions about the current or future state of the world such as what is the probability that a severe storm will hit Barnacle? Or what is the probability that Bus_1 will get another flat tire? Answers to these questions can be used to improve the plan. For example, we might send storm supplies to Barnacle or send Bus_1 on a longer but better paved road. We might also ask whether the plan will have a higher chance of succeeding if we perform some action. For example, given the evidence received so far, will the plan have a higher chance of succeeding if we send a helicopter to evacuate the people in Exodus compared to letting Bus_3 do it, as originally planned.

We cast plan monitoring as inference on Bayesian networks, but we interested in planning domains for which the resulting networks are difficult to solve with exact methods. Simulation is a promising alternative when analyzing large plans in uncertain domains (e.g., (Lesh *et al.* 1998)). Simulation is linear in the length of the plan and fast in practice. Furthermore, the number of simulations required to estimate the probability of an event to some level of confidence is independent of the length of the plan.

In this paper, we explore the use of simulation for performing plan monitoring. Our objective is an algorithm for quickly generating a set of weighted simulations, where the weight indicates the probability of the simulation given observations made during the partial execution of a plan.

There are several algorithms for simulation-based inference in Bayesian networks, including likelihood weighting (Shachter and Peot 1989; Fung and Chang. 1989), arc reversal (Shachter 1986; Fung and Chang. 1989), and Survival Of The Fittest (SOF) (Kanazawa *et al.* 1995). We consider these algorithms and find that none of them is especially well suited for (or was designed for) plan monitoring. We then present a modification to SOF that improves its performance and introduce a new algorithm, Rewind/Replay (RR).

In the remainder of this paper, we formulate our task, discuss previous algorithms, and consider two problems on which past approaches perform poorly. We then present RR and a generalization of SOF. Finally, we describe our experiments and conclude.

2 Problem formulation

Previous work shows how probabilistic processes and plans can be encoded in Bayesian networks (e.g., (Dean and Kanazawa 1989; Hanks *et al.* 1995)). Here, we formulate our task as inference on Bayesian networks.

A Bayesian network describes the joint distribution over a finite set of discrete random variables $\mathcal{X}$. Each variable $X_i \in \mathcal{X}$ can take on any value from a finite domain $val(X_i)$. A *variable assignment* is an expression of the form $X_i = x_j$ indicating that X_i has value x_j. We use capital letters (e.g., X, Y, Z) for variables and lowercase letters (e.g., x, y, z) for values of variables.

A Bayesian network is a directed, acyclic graph in which nodes correspond to variables and arcs to direct probabilistic dependence relations among variables. A network is defined by a variable set $\mathcal{X}$, a parent function Π, and a conditional probability table CPT. The Π function maps X_i to X_i's parents. The CPT function maps X_i and a variable assignment for each parent of X_i to a probability distribution over $val(X_i)$. Variable X_i can be *sampled* by randomly drawing from the probability distribution returned by CPT. An entire network can be sampled by sampling the variables in an order such that each variable is sampled after its parents are sampled. See (Pearl 1988) for details.

Although we formulate our problem more generally, we are especially interested in *Dynamic belief networks* (DBNs), also called temporal belief networks (e.g., (Kjaerulff 1992)). DBNs are Bayesian networks used to model temporal processes. A DBN can be divided into subsections, called *time slices*, that correspond to snapshots of the state of the world.

Typically, certain variables in each time slice are designated as observation variables. Figure 1 shows a very simple DBN. The state variables Pos_i and Dir_i represent a person's position and direction, respectively, at time i. The person's direction at time i influences both their position and direction at time $i+1$. The variable Obs_i represents some observation on Dir_i, such as the output of a compass or the person calling out their direction. The observations may be noisy. Note that we can infer a probability distribution over the position of the person given a set of observations, even though the position is never directly observed.

We now define the *simulated inference task*. The inputs are a Bayesian network B and a set of assignments $\mathcal{O} = \{O_1 = o_1, ..., O_n = o_n\}$, which give the observed values for some of the observation variables in B, such as $\{Obs_1 = west, Obs_2 = east\}$. The output is a set of positively weighted samples of B, where each sample s_i is weighted by an estimate of the probability $\mathbf{P}(s_i|\mathcal{O})$.

Inference on Bayesian networks is more typically formulated as determining the probability of a query expression given the evidence. We chose our formulation because it offers a wide range of reasoning for plan monitoring. Suppose we are projecting from past observations to find the safest escape route in some hazardous environment. We might query a probabilistic reasoner

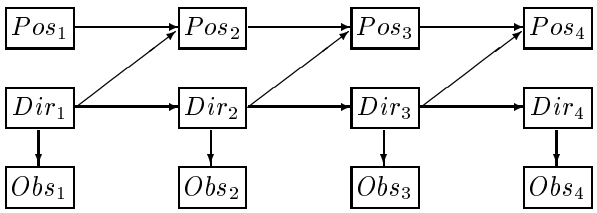


Figure 1: Example DBN

to infer the probability that each possible path is safe, but this would mean enumerating all possible paths. As an alternative, we can apply path-following algorithms to a set of weighted simulations and more quickly determine the safest path. We give an example of this approach in our experiments.

3 Previous algorithms

We now discuss previous simulation methods.

3.1 Logical sampling

A simple approach, called *logical sampling* (LS) (Henrion 1988), is to repeatedly sample the network and discard samples in which $O_i \neq o_i$ for any $O_i = o_i \in \mathcal{O}$. LS assigns the same weight to each retained simulation. Given a query Ω, LS estimates the probability of Ω as the percentage of retained samples in which Ω holds. Logical sampling is an *unbiased* technique: as the number of simulations approaches infinity, the estimate of Ω approaches the true probability of Ω.

The probability of a perfect match, however, is exponentially low in the number of observations, assuming some independence among the observations. In many examples we consider, LS can run for hours without retaining a single sample.

3.2 Likelihood weighting

The most commonly implemented simulation technique, called *likelihood weighting* (LW), is a variation on logical sampling which can converge much faster in some cases (Shachter and Peot 1989; Fung and Chang. 1989; Dagum and Pradhan 1996).

LW also samples the network repeatedly but weights each instantiation by the probability it could have produced the observations. Consider a simple case in which there is a single observation node O_1 with observed value o_1. Logical sampling would repeatedly simulate the network and discard any samples in which $O_1 \neq o_1$. LW, however, weights each sample by the probability that O_1 would be assigned o_1 under the network's CPT function given the values of O_1's parents in the sample.[1] For example, if O_1 had a .08 probability of being assigned o_1 but was assigned some other value, LW would weight the sample .08.

[1]We assume observation variables have no children. Thus, their values do not effect the value of other variables.

In the more general case, LW weights each sample s as the Likelihood($\mathcal{O}|s$), where **Likelihood**($\mathcal{O}|s$) denotes the product of the individual conditional probabilities for the evidence in $\mathcal{O}$ given the sampled values for their parents in s.

LW is also unbiased and has been shown to be effective on many problems. However, as demonstrated in (Kanazawa *et al.* 1995), LW does not work well on DBNs. Both LS and LW consider the evidence only when weighting the samples. When the network models a temporal process with an exponential number of possible execution paths, the vast majority of random samples can have a likelihood of or near zero.

3.3 SOF

The *Survival Of The Fittest* (SOF) algorithm is the best suited, of those we investigated, for plan monitoring because it uses the evidence to guide the simulation, rather than simply to weight it afterwards (Kanazawa *et al.* 1995). We now provide an informal description of the SOF algorithm. Figure 5 contains pseudo-code for a slight generalization.

SOF was designed specifically for DBN's. The idea is to, at each time point, seed the next round of simulations with the samples that best matched the evidence in the previous round. SOF maintains a fixed number of possible world states, but re-generates a sample population of world states for each time t by a weighted random selection from the world states at time $t-1$ where the weight for a world state is given by the likelihood of the observations at time t in that world state.

Initially, SOF generates a fixed number, say 100, of samples of the state variables in the first time slice. The weight of the ith sample, s_i, is the likelihood of the observed evidence at time 1. SOF now randomly generates a new set of 100 samples by randomly selecting from samples $s_1, ..., s_{100}$ using weights $w_1, ..., w_{100}$. Some samples may be chosen (and copied) multiple times and others may not be chosen at all. SOF next samples the state variables in the second slice in each of its 100 samples. SOF then weights each of these samples by the likelihood of the evidence at time 2. Next, SOF re-populates the samples again, weights them by the likelihood of the evidence at time 3, and so on, until all the evidence has been accounted for.

4 Difficult problems for SOF

We now describe two example DBN's carefully designed to expose potential problems with using SOF for plan monitoring.

4.1 Discarding plausible hypotheses

SOF can discard hypotheses by random chance during its re-population phase. Consider the following thought experiment. Suppose you have 1,000 distinct names in a hat and you repeatedly draw a name from the hat, write it down, and return the name to the hat. If you repeat the process 1,000 times you will have, on average,

about 632 distinct names.[2] SOF can lose hypotheses in a similar manner. Even though the evidence favors selecting the most likely hypotheses, the problem will arise if there are hidden state variables whose value becomes apparent only by integrating information over time from a series of reports.

We designed the following network to demonstrate this problem. The network contains one state and one sensor node for each time slice. At time 1, the state node is assigned a integer value between 1 and 50 from a uniform distribution. For $t > 1$, the state simply inherits its value from the state node at time $t-1$. At each time step $t \geq 1$, if the state node's value is evenly divisible by t then the sensor returns *Yes* with .9 probability or *No* with .1 probability and otherwise returns *Yes* with .1 probability or *No* with .9 probability. If, for example, we observe Yes, Yes, Yes, No, Yes, then there is a 22.5% chance that the state's value is 30.

The goal is to guess the state node's value. This seems like an easy problem for SOF to solve with, say, 1000 samples. There are only 50 distinct hypotheses and thus SOF should initially have several copies of each hypotheses. As time progresses, the most likely hypotheses should be steadily re-enforced by the evidence. We tested SOF by running it with $N = 1000$ samples. We found the hypothesis that SOF reports is most likely and see if it is, in fact, one of the most likely hypotheses given the evidence.[3] Based on 1,000 example trials, however, SOF achieves only an 81% accuracy on this problem. Even with $N = 2000$, SOF achieves only 92% accuracy.

The graph in figure 2 shows the average number of distinct hypotheses that SOF, with $N = 1000$, maintains after t time steps for $0 \leq t \leq 25$ as well as the average number of hypotheses that have a .005 or greater probability of being true given the evidence at time t.[4] We counted the number of hypotheses maintained by SOF by counting the number of distinct values for the state node contained in the 1000 samples maintained by SOF. As the graph shows, the number of hypotheses that SOF maintains is noticeably lower than the number of plausible hypotheses.

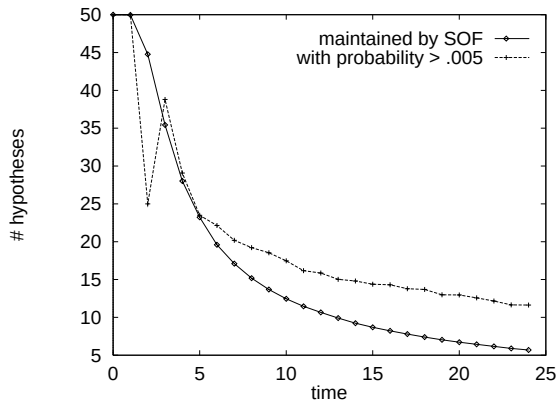


Figure 2: Number of hypotheses maintained by SOF compared to the number of hypotheses with probability greater than .005 on the NumberNet DBN.

4.2 Premature variable assignment

We designed the following problem to reduce the usefulness of SOF's re-population strategy. We consider a case in which the state variables need to be assigned values several time steps before they are observed. We believe this represents an important aspect of plan monitoring. Probabilistic planners (e.g. (Kushmerick *et al.* 1995)) take as input a probability distribution over initial states. The value of conditions in the initial state are decided before they are observed, but may be crucial to the success of a plan. For example, in evacuation plans, there might be uncertainty about whether or not a certain bridge is usable, and the first report on its condition may come when a vehicle encounters it during plan execution.

Consider a simple DBN in which there are K fair coins, all flipped at time 1. The ith coin is reported with .95% accuracy at time $i+1$ and again with with .95% accuracy at time $K+i+1$. The network for this DBN contains a state node c_i for each of the K coins and a single sensor node. Initially all the coins are given a random value from $\{Heads, Tails\}$. For all times $j > 1$ the coin c_i simply inherits its value at time $j-1$. At time $j+1$ the sensor outputs the value of coin c_j at time $j+1$ with .95 accuracy. Similarly, at time $K+j+1$ the sensor outputs the value of coin c_j at time $K+j+1$ with .95 accuracy.

Given a sequence of observations, the goal is compute the probability that coin $c_i = Heads$ for each coin i. To evaluate SOF, we computed the actual probability of $c_i = Heads$ given the evidence and compared it to the weighted estimate produced by diving the sum of the weights of the simulations SOF returns in which $c_i = Heads$ by the sum of the weights of all returned simulations.

The graph in figure 3 shows the average error rate on SOF on a problem with 15 coins if SOF maintains $N = 1000$ samples. The graph shows both that the errors are high (the average error is .24) and that the error is higher for the higher number coins. Our explanation is as follows. At time 1, SOF picks 1000 of the possible 2^{15} combinations of *Heads* or *Tails* for the coins. At time 2, SOF re-populates its samples to match the first report on coin 1. At time 3, SOF does the same for coin 1. By the time SOF gets to the higher number coins, it less and less likely to have samples with a coin flip

[2] Each name has a 1 - $.999^{1000}$ chance of being selected.

[3] It is simple to compute the actual probability of the observed evidence for each of the 50 (equally likely) hypotheses, but there might be ties.

[4] The number of hypotheses with probability $\geq .005$ can increase. At time 2, half the hypotheses are relatively unlikely compared to the other half. At time 3, typically, a few are very likely but a larger number become reasonably likely.

matching the observed evidence.

Figure 3 also shows the performance on a modified version of SOF, called SOF-bayes, which we describe below. As shown in the graph, SOF-bayes performs much better on this problem.

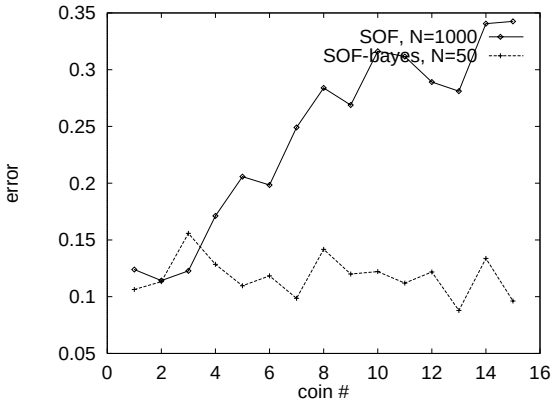


Figure 3: Error rate for SOF with $N = 1000$ on the CoinNet problem, compared with our modified version of SOF, called SOF-bayes, with $N = 50$. Based on 200 random trials.

5 New algorithms

We now present new methods.

5.1 Rewind/replay algorithm

We now introduce the *Rewind/Replay* (RR) algorithm, which is closely related to the sampling method called sequential imputation (discussed below).

We first describe RR informally in terms of solving a DBN. RR samples each time slice of the network a fixed number of times, before moving onto the next time slice. After each round of sampling the nodes in time slice i, RR adopts one of the samples by choosing at random from the samples which contain all the observations made at time i. It then simulates forward from this sample into the $(i+1)$th time slice. If the observed evidence does not arise in any of the samples of a time slice, then RR abandons the current sample of the DBN that it is constructing, and begins again from the first time slice. If RR makes it through all the time slices then it has a single sample of the network that matches all the evidence. RR weights this sample as $F_1 \times F_2 \times ..F_n$ where F_i is the fraction of samples of time slice i that matched the observations for time i. The justification for this weighting method is that the probability of the sample given the evidence is $\mathbf{P}(E_1 \mid S_1) \times \mathbf{P}(E_2 \mid S_2) \times ... \times \mathbf{P}(E_N \mid S_N)$, where $\mathbf{P}(E_i \mid S_i)$ is the probability of the evidence at time i given the state at time i and is approximated by F_i.

An optimization we use in our implementation of RR is to proceed to the next time slice as soon as the evidence arises in any sample. This is as random as choosing from the samples that matched the evidence after R_{max} tries, since all the trials are independent. If RR manages to match the evidence for all the time slices, it can compute the weight by returning to each time slice j and perform $R_{max} - k_j$ samples where k_j is the number of samples RR has already performed on the jth time slice. The advantage here is to reduce the computation expended in RR's attempts to sample the network that ultimately fail.

The intuition behind RR can be explained in terms of the task of flipping twenty coins until all of them are heads. The logical sampling approach, for example, is to flip all twenty coins, and if they do not all come up heads, flip them all again. This is expected to succeed in 1 out of 2^{20} trials. The Rewind/Replay approach is to flip each coin until it is heads and then move on to the next coin. For $R_{max} = 5$, RR should succeed with $1 - (\frac{2^5-1}{2^5})^{20}$ probability, or about 1 in 2 times. With $R_{max} = 10$, RR will succeed with over .99 probability.

We now describe RR in terms of an arbitrary Bayesian network. To do so, we need to distribute the input observations $\mathcal{O} = \{O_1 = o_1, ..., O_n = o_n\}$ into a sequence of sets of observations $\mathcal{E}_1, \mathcal{E}_2, ...\mathcal{E}_m$ such that each observation goes into exactly one evidence set. Although it will not effect correctness, the distribution of the evidence can have a significant impact on the performance of RR. For example, if all the evidence is put into a single set, i.e, $\mathcal{E}_1 = \{\mathcal{O}\}$, then RR is equivalent to LS. Unless otherwise stated, we will assume the opposite extreme of putting each observation in its own set, i.e, for $1 \leq i \leq n$, $\mathcal{E}_i = \{O_i = o_i\}$.

RR needs to determine which nodes influence the observations in each set $\mathcal{E}_j$. In a Bayesian network, the only nodes which can influence the probability distribution over the value of some node O_i when O_i is sampled are the ancestors of O_i (i.e., the parents of O_i, the parents of the parents O_i, and so on). As we process each evidence set $\mathcal{E}_j$, we want to sample only the variables which are ancestors of $\mathcal{E}_j$ and that we have not sampled already, i.e., that are not ancestors of any observation in $\mathcal{E}_1, ..., \mathcal{E}_{j-1}$. This set is determined by the function ANC, defined in figure 4.

The pseudo code for RR is shown in figure 4. For each simulation, RR iterates through the evidence sets $\mathcal{E}_1$,..., $\mathcal{E}_m$. For each $\mathcal{E}_j$, RR samples the unsampled ancestors of $\mathcal{E}_j$ a total of R_{max} times and selects one the samples that satisfies every variable assignment in $\mathcal{E}_j$. The weight for each net is $\frac{M_1}{R_{max}} \times \frac{M_1}{R_{max}} \times ... \times \frac{M_m}{R_{max}}$, where M_1 is the number of samples that matched evidence set $\mathcal{E}_i$.

The RR algorithm is closely related to the statistical sampling technique called sequential imputation (Kong *et al.* 1994; Liu 1996). Sequential imputation is a Monte Carlo technique for solving nonparametric Bayes models given some data $d_1, ..., d_i$ by incrementally sampling from the probability distribution $\mathbf{P}(d_i \mid d_1, ..., d_{i-1})$ for increasing values of i. RR is perhaps best viewed as one of many possible instantiations of sequential imputation for the task of infer-

procedure: RR$(B, \mathcal{E} = \{\mathcal{E}_1 ... \mathcal{E}_m\}, N, R_{max})$
for $i = 1$ to N
  $s_i \leftarrow \emptyset$; $w_i \leftarrow 1$; $stop \leftarrow false$
  for $j = 1$ **to** m
    if $stop = false$
      $X_j \leftarrow$ ANC$(\mathcal{E}_j, \mathcal{E}, B)$
      $\mathcal{M} \leftarrow \emptyset$
      for $l = 1$ **to** R_{max}
        Add a sample of X_j to s_i
        if HOLDS$(\mathcal{E}_j, s_i)$
          Add a copy of s_i to $\mathcal{M}$
      $w_i \leftarrow w_i \times \frac{|\mathcal{M}|}{R_{max}}$
      if $|\mathcal{M}| > 0$
        $s_i \leftarrow$ random selection from $\mathcal{M}$
      else $stop \leftarrow true$
return $s_1, ... s_N$ and $w_1, ..., w_n$.

function: ANC$(\mathcal{E}_j, \{\mathcal{E}_1 ... \mathcal{E}_m\}, B)$
  Returns set of all variables V_i in B from which there is a directed path in B from V_i to a variable in $\mathcal{E}_j$ but not a directed path to any variable $e_j \in \mathcal{E}_1 \cup \mathcal{E}_2 \cup ... \cup \mathcal{E}_{j-1}$.

Figure 4: The Rewind/Replay algorithm and support function ANC

ence on Bayesian networks, which employs a particular method of extending the sample to cover the ith piece of evidence. Likewise, SOF can be thought of as an alternative instantiation of sequential imputation.

RR uses the evidence to guide the simulations, but can avoid the problems caused by SOF's re-population phase. With $R_{max} = 10$ and $N = 500$, RR achieved .96 accuracy on the NumberNet problem described above, on 1000 random trials. In the worst case, RR might effectively sample the network $10 \times 500 = 5000$ times, but in practice RR samples it much less frequently and, on average, only generates 69.4 matches per 500 attempts. In our Lisp implementations, RR required 3.3 CPU seconds to complete its inference, which is comparable to the 4.8 CPU seconds for SOF with $N = 1000$ to achieve an accuracy of .82. Additionally, RR achieved .09 error on the CoinNet problem, with $R_{max} = 10$ and $N = 1,000$.

5.2 SOF-bayes

Figure 5 shows pseudo code for a variation of SOF that works on an arbitrary Bayesian network. Rather than sample the nodes time slice by time slice, as SOF does, SOF-bayes distributes the evidence into evidence sets $\mathcal{E}_1, ..., \mathcal{E}_n$, as described above for RR, and samples the unsampled ancestors of each $\mathcal{E}_i$ before re-populating.

Compared to SOF, SOF-bayes lazily samples the network, only assigning variables to state nodes when forced to in order to account for some observed evidence. In the CoinNet problem, for example, SOF-bayes does not sample the value of a coin until it is observed and thus generates N samples of that coin's

procedure: SOF-bayes $(B, \mathcal{E} = \{\mathcal{E}_1 ... \mathcal{E}_m\}, N)$
for $i = 1$ to N
  $s_{1,i} \leftarrow \emptyset$; $w_i \leftarrow 1$
for $j = 1$ **to** m
  $X_j \leftarrow$ ANC$(\mathcal{E}_j, \mathcal{E}, B)$
  for $i = 1$ **to** N
    $s_{j,i} \leftarrow$ randomized selection from $s_{j-1,1}, ..., s_{j-1,N}$
      weighted by $w_1, ..., w_N$.
    Add a sample of X_j to $s_{j,i}$
    $w_i \leftarrow$ LIKELIHOOD$(E_j | s_i)$
return $s_{m,1}, ... s_{m,N}$ and $w_1, ..., w_n$.

Figure 5: The Survival-of-the-fittest algorithm for an arbitrary Bayesian network

value at that time. In contrast, SOF generates N samples of the coin's value in the first time step and may discard most of those samples by the time the coin's value is observed. As shown in figure 3, SOF-bayes performs significantly better than SOF on the CoinNet problem.

Another advantage of SOF-bayes is that it does not have to sample the entire time slice at once. Consider a case where K coins are flipped and reported in each time slice. SOF needs 2^K samples in order to generate a single sample matching all the coin flips. SOF-bayes can, however, match each coin flip individually.

6 Experiments

We now describe our experiments. Our goal was to generate a large, non-trivial DBN in order to demonstrate the promise of simulation-based reasoning for plan monitoring in uncertain domains.

6.1 Description of fire domain

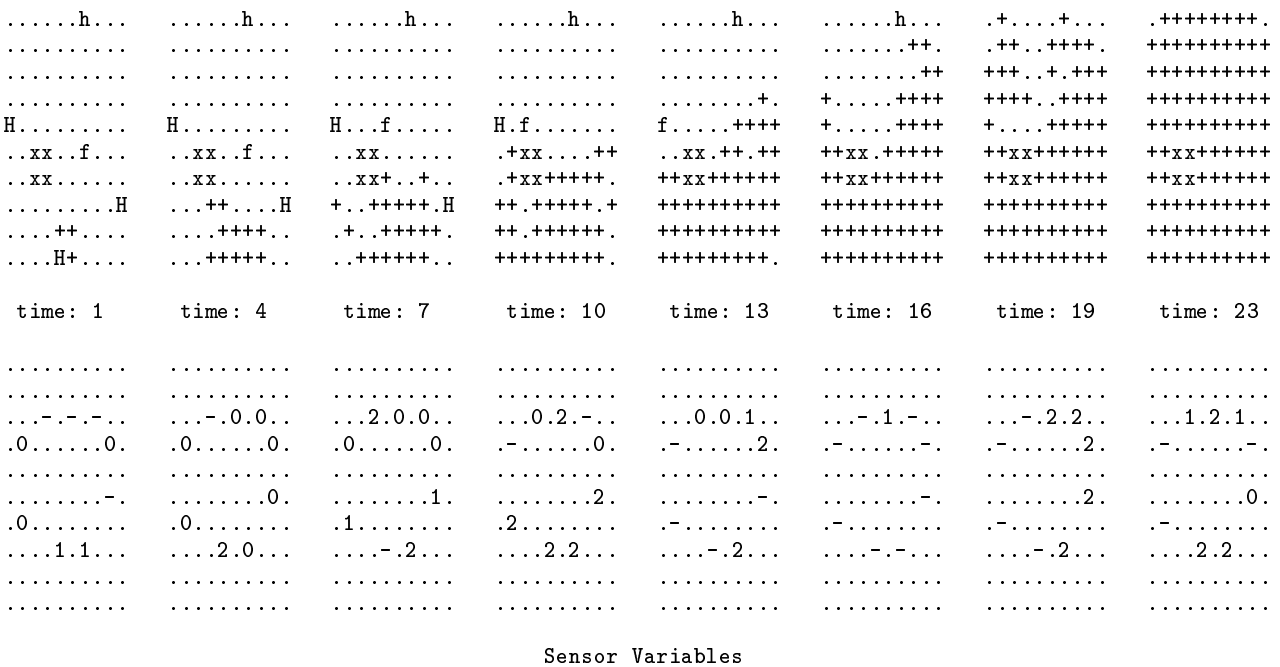
We constructed a simple forest-fire domain based loosely on the Phoenix fire simulator (Hart and Cohen 1992), which in turn is based on model of fire spread from the National Wildlife Coordinating Group Fireline Handbook (Group 1985).

We use a grid representation of the terrain. During the course of the simulation, each cell is either burning or not. At each time step, the probability that a cell will be burning is computed (by an externally defined function) based on the status of that cell and that cell's neighbors at the previous time slice.

There are ten noisy sensors, each of which reports on a 3×3 block of grid cells. The sensor can output *Low*, *Med*, or *High*, indicating that 0, 1–3, or 4–9 of the cells are on fire, respectively (with 10% noise added in). Additionally, each sensor has a solar battery with a 75% chance of being charged in each time slice. If the battery is not charged at time t then the sensor outputs *Recharging* at time $t + 1$.

A fire fighter is initially situated in the middle of the grid. The fire fighter monitors the fire sensors in order to decide whether or not to flee to one of four helipads situated on the edge of the grid. Each helipad,

State Variables



Sensor Variables

Figure 6: ASCII representation of several time slices of an example simulation of the fire world domain. A '+' indicates fire, an 'H' indicates a working helipad, an 'h' indicates a non-working 'helipad', an 'x' indicates an impassable terrain. The fire fighter is denoted by an 'f'. In the sensor readings, a 0 indicates a low reading, a 1 indicates, a medium reading, a 2 indicates a high reading, and a '-' indicates that the sensor is recharging. The messages indicating the condition of the helipads are not shown here.

however, has only a .65 chance of being functional. The fire fighter receives reports of varying reliability about the condition of four helipads.

In these experiments, we used a 10×10 grid, and created a DBN with 30 identical time slices, resulting in a network with 3660 nodes. Most nodes in the network have nine parents. There are a total of 14 observations per time slice (ten fire sensors and four helipad reports.) A complete description and simulation traces for the domain are available by email from the first author. Figure 6 contains an ASCII representation of several time slices from one execution trace.

6.2 Results

The first set of experiments we ran were to determine how accurately the algorithms could predict aspects of the world state from past observations. The results in table 1 were generated as follows. First we generated a sample of the network, s_r. We then set $\mathcal{O}_r$ to be the variable assignments for all the sensor nodes in s_r in the first 6 time steps. We then called LW, SOF, SOF-bayes, and RR with observations $\mathcal{O}_r$. The algorithms ran until they produced at least 50 sample simulations or a 3 minute time limit elapsed (although we do not interrupt the algorithm when the time limit elapses). We then evaluated the sample simulations returned by the inference algorithms against the sample s_r. For each grid cell, we compared the status (*Burning* or *Safe*) of the cell at time 10 with the weighted estimate of the returned cell. We credit the algorithm with a correct prediction if the weighted estimate of the cell being *Safe* was greater than .5 and the cell's status in s_r is *Safe* or if the weighted estimate is less than .5 and the cell's status is *Burning*. Similarly, for each helipad, we compared value of the helipad in s_r (either *Usable* or *Unusable*) with the weighted estimate of the returned distributions. If no distributions were returned, we counted this as guessing that all cells are *Safe* and all helipads *Usable*. As shown in table 1, RR performs best, and SOF-bayes performed better than SOF. LW never generated any matches, primarily because it never matched all the *Recharging* signals, and thus we did not list any accuracies for LW.

In the second set of experiments, we used the weighted simulations to control the movements of the fire fighter and then compared the survival rate with the different inference algorithms. In each iteration the fire fighter can stay in its current cell or move to one

	LW	SOF $N = 1000$	SOF-bayes $N = 1000$	RR $R_{max} = 50$
CPU seconds	181	235	168	128
Accuracy predicting if cell contains fire	–	.64	.83	.82
Accuracy predicting condition of helipads	–	.53	.66	.94

Table 1: Accuracy at predicting fire and the condition of the helipads. Numbers averaged over 100 trial runs.

	LW	SOF $N = 1000$	SOF-bayes $N = 1000$	RR $R_{max} = 75$
Survival rate	–	.61	.75	.95

Table 2: Survival rate of the simulated fire fighter using different inference engines. Numbers averaged over 25 trial runs.

of the eight adjacent cells. The procedure for deciding where to go computes, for each weighted simulation and each cell, whether or not there is a path to safety from the cell. Based on the weights of the simulations, the procedure then moves to one of the cells with the highest probability of being safe, with a preference for remaining in the same cell. To speed the experiments, we only re-computed the weighted simulations given the observations at time 6, 8, and 10. As shown in table 2, the fire fighter survives much more often when using the Rewind/Replay algorithm than any of the other algorithms.

7 Related work and conclusions

One previous technique we have not mentioned yet is arc reversal (Shachter 1986; Fung and Chang. 1989), which has been applied to DBNs (Cheuk and Boutilier 1997). Arc reversal involves reversing the arcs that point to the evidence nodes. This technique essentially uses the information in the CPT of the Bayesian network to perform reverse simulation from the observed evidence. Arc reversal can increase the size of the network exponentially and would so on our networks. When reversing an arc from node n_1 to node n_2, all parents of n_1 and n_2 become parents to both nodes. Reversing a single arc in the fire network can increase the number of parents of a node from 9 to 17. This increases the size of the CPT for that node from 2^9 to 2^{17}. Furthermore, in our algorithm the original CPT is encoded compactly as a function but would have to be expanded out into a table to reverse an arc.

There have been many other approaches to approximate inference on Bayesian networks (e.g., (Dechter and Rish 1997)) though most have not been specifically evaluated on complex temporal processes. (Boyen and Koller 1998) proposes a method for approximating the belief state for DBNs and prove that under certain assumptions, the error in the belief state will contract exponentially over time. When applied to DBNs, the approach involves grouping the variables in each time slice into semi-independent subsets and maintaining independent approximations of each of these groupings. It is not clear, however, how to cluster the variables representing the cells in our fire world network into manageable subsets because each cell is strongly connected to its neighbors, which in turn are strongly connected to all their neighbors. Finally, prior work has suggested the use of sequential imputation for solving Bayes nets (Hanks *et al.* 1995), but did not propose the specific Rewind/Replay implementation of this idea or compare RR to SOF.

In this paper we have investigated simulation techniques for plan monitoring and shown that some techniques have significant advantages over others. We have extended existing simulation algorithms by generalizing SOF to arbitrary bayes networks and introducing the Rewind/Replay algorithm.

References

X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. 14th Conf. Uncertainty in Artifical Intelligence*, pages 33–42, 1998.

A. Y.W Cheuk and C. Boutilier. Structured arc reversal and simulation of dynamic probabilistic networks. In *Proc. 13th Conf. Uncertainty in Artifical Intelligence*, pages 72–79, 1997.

P. Dagum and M. Pradhan. Optimal Monte Carlo Estimation Of Belief Network Inference. In *Proc. 12th Conf. Uncertainty in Artifical Intelligence*, pages 446–453, 1996.

Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.

R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *Proc. 13th Conf. Uncertainty in Artifical Intelligence*, 1997.

R. Fung and K Chang. Weighing and integrating evidence for stochastic simulation in bayesian networks. In *Proc. 5th Conf. Uncertainty in Artifical Intelligence*, pages 209–219, 1989.

National Wildlife Coordinating Group. *NWCG Fireline Handbook*. Boise, Idaho, November 1985.

S. Hanks, D. Madigan, and J. Gavrin. Probabilistic temporal reasoning with endogenous change. In *Proc. 11th Conf. Uncertainty in Artifical Intelligence*, pages 245–254, 1995.

D.M Hart and P.R Cohen. Predicting and explaining success and task duration in the phoenix planner. In *Proceedings of the First International Conference on AI Planning Systems*, pages 106–115, 1992.

M. Henrion. Propagation of ncertainty in bayesian networks by probabilistic logic sampling. In *Uncertainty in Artificial Intelligence 2*, pages 149–163, 1988.

K. Kanazawa, Koller D., and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *Proc. 11th Conf. Uncertainty in Artifical Intelligence*, pages 346–351, 1995.

U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proc. 8th Conf. Uncertainty in Artifical Intelligence*, pages 121–129, July 1992.

A. Kong, J.S. Liu, and W.H. Wong. Sequential imputation and bayesian missing data problems. *J. Americian Statistical Association*, pages 278–288, 1994.

N. Kushmerick, S. Hanks, and D. Weld. An Algorithm for Probabilistic Planning. *J. Artificial Intelligence*, 76:239–286, 1995.

N. Lesh, N. Martin, and J. Allen. Improving big plans. In *Proc. 15th Nat. Conf. AI*, pages 860–867, 1998.

J.S. Liu. Nonparametric hierarchical bayes via sequential imputation. *Ann. Statist.*, pages 910–930, 1996.

J. Pearl. *Probablistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo, CA, 1988.

R.D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Proc. 5th Conf. Uncertainty in Artifical Intelligence*, 1989.

R.D. Shachter. Evaluating influence diagrams. *Operations Research*, 33(6):871–882, 1986.